AFIT/GE/ENG/97D-17

SPEAKER VERIFICATION IN THE PRESENCE OF CHANNEL MISMATCH USING GAUSSIAN MIXTURE MODELS

THESIS

Robert Brian Reid
Captain, USAF

AFIT/GE/ENG/97D-17

AFIT/GE/ENG/97D-17

SPEAKER VERIFICATION IN THE PRESENCE OF CHANNEL MISMATCH USING GAUSSIAN MIXTURE MODELS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Robert Brian Reid, M.B.A., B.S.E.E.
Captain, USAF

December, 1997

AFIT/GE/ENG/97D-17

SPEAKER VERIFICATION IN THE PRESENCE OF CHANNEL MISMATCH USING GAUSSIAN MIXTURE MODELS

Robert Brian Reid, M.B.A., B.S.E.E.

Captain, USAF

Approved:

[illegible]	20 Nov 97
Dr. Martin P. DeSimio Thesis Advisor	Date
Timothy R Anderson	20 Nov 97
Dr. Timothy R. Anderson Committee Member	Date
Raymond E. Slyh	20 Nov 97
Dr. Raymond E. Slyh Committee Member	Date

Acknowledgements

No undertaking of lasting value is ever accomplished alone. While only time will determine the worth of this project to the world, it has been a tremendous learning experience for me. I would be remiss if I failed to thanks those who have had a hand in helping bring this research out of my notebook and into print. First, I would like to thank my fellow classmates for their help in getting me under way using yet another word processing environment: Todd Hale and his trail blazing efforts on style files and formats and Jim Louthain and John Fitton for their assistance with LaTeX. Next, my thanks go to Al Harb and his willingness to share his insights in handling the TIMIT and NTIMIT data. I appreciated the time and resources provided by Capt John Colombi, Ph.D., through the National Security Agency without which this research would have gone much slower and on a much smaller scope. I would also like to thank the members of my thesis committee for their help over the last 18 months: Dr. Timothy Anderson and his enormously valuable inputs on using HTK; and Dr. Raymond Slyh and his diligence in "fine-tuning" this work. A heart felt thanks to my thesis advisor, Dr. DeSimio, for his patience and guidance during this endeavor. For Amy, my companion, friend, and wife, and her enormous patience, understanding and support during this endeavor – thank you for helping me keep things in perspective. Without you this would have been a much heavier load. Most importantly, praise to the one who is the Alpha and Omega, who provides the strength and grace to meet each day, without whom this undertaking would have been impossible.

Robert Brian Reid

Table of Contents

List of Figures

List of Tables

AFIT/GE/ENG/97D-17

Abstract

A channel compensation method is sought for use in speaker identification (ID) and verification applications under matched and mismatched training and testing conditions. This work expands on previous work on matched conditions by investigating three techniques on matched and mismatched conditions using the TIMIT and NTIMIT speech databases. First, previous results on 168 speakers are reproduced for matched conditions using Gaussian mixture models (GMM) and mel-frequency cepstral coefficients. Next, cepstral mean subtraction with band limiting (CMSBL) is investigated. The third method, developed in this thesis, uses a modified Wiener filtering approach to channel compensation. New GMMs are created for each method. The first approach is then expanded to include all 630 TIMIT and NTIMIT speakers for speaker verification. For speaker ID under matched conditions, the CMSBL method had three more errors than no additional preprocessing but yielded the best ID results for the mismatch case with 27.4% correct. Additionally, the CMSBL method yielded the best verification results with an equal error rate of approximately 0.26% for matched conditions on TIMIT and approximately 19.6% for mismatched conditions on NTIMIT.

SPEAKER VERIFICATION IN THE PRESENCE OF CHANNEL MISMATCH USING GAUSSIAN MIXTURE MODELS

I. Introduction

1.1 Background

What is speaker verification in the presence of channel mismatch? Speaker verification is related to the process of speaker identification (ID), also known as speaker recognition [6], where a machine attempts to determine which speaker, out of a group of registered speakers, a recorded portion of speech came from. Verification, on the other hand, starts with a person claiming a particular identity. The machine must then determine whether or not the speaker is who they claim to be. Proper verification is critical for controlling access to sensitive information or special areas. Just as people have difficulty identifying others over the telephone, computers have difficulty correctly identifying or verifying people when speech is recorded differently than what the computer was trained with. This difference in training and testing conditions is channel mismatch. While Gaussian mixture models using Mel-frequency cepstral coefficients have had considerable success for similar training and testing conditions [14], currently no method adequately handles the cases where training and testing conditions are different. Consequently, a solution for speaker verification in the presence of channel mismatch is needed.

1.2 Problem Statement

Develop a channel compensation method for speaker identification (ID) and speaker verification that compensates for the channel mismatch between training and testing conditions.

1.3 Assumptions

In order to facilitate development, this thesis assumes that all training collections are cooperative. For this thesis, cooperative means that a speaker's voice was recorded with their knowledge under controlled conditions, and that the speaker made no conscious attempt to alter their voice to sound like one of the other speakers or someone other than themselves. For telephone quality speech, the utterance passed to the system was modified only by the telephone channel. Due to the nature of the databases, a subject is assumed to be in the same physical and emotional states during training and testing.

1.4 Scope and Research Objectives

The TIMIT and NTIMIT databases are used to compare and analyze previous spectral processing methods to a new filtering approach. The ability of processing methods for speaker ID and verification in channel mismatch conditions will be experimentally evaluated. Towards that end, the following are the desired research objectives:

- Reproduce previous speaker ID results using Gaussian mixture models (GMMs) on TIMIT and NTIMIT
- Reproduce previous speaker verification results using GMMs under matched conditions using TIMIT and under channel mismatch conditions using NTIMIT
- Extend previous speaker verification results to a larger population
- Experimentally evaluate the effect of a modified Wiener filter preprocessor for speaker ID and verification tasks on TIMIT and NTIMIT

1.5 Organization

The remainder of the thesis is divided into four chapters. Chapter II presents the theory behind using Mel-frequency cepstral coefficients (MFCCs) and Gaussian mixture models (GMMs) when performing speaker identification and speaker verification. Chapter II also outlines the channel compensation techniques of cepstral mean subtraction and the modified Wiener filtering technique. Chapter III outlines the computer equipment and software and how they were used in calculating the results. Chapter IV presents the results of the thesis broken down by identification or verification and then further subdivided by corpus and method. Chapter V highlights the conclusions drawn from the research and includes suggestions for future study.

II. Theory

2.1 Introduction

This chapter outlines the theory behind speaker identification and verification using Mel-frequency cepstral coefficients (MFCCs) and Gaussian mixture models (GMMs). First, motivation for using Mel-frequency cepstral coefficients is provided, and then the basics of how to generate them is outlined. Next, the theory behind Gaussian mixture models and their training is explained. The following sections then explain the theory for speaker identification (ID) and the steps involved for speaker verification. The chapter also outlines the theory behind two channel compensation techniques, the commonly used cepstral mean subtraction [6,9,15] and a new modified Wiener filter approach.

2.2 Mel-frequency Cepstral Coefficients (MFCCs)

2.2.1 Motivation for using MFCCs. MFCCs provide a compact representation for modeling an individual's vocal tract by separating it from the pitch information through homomorphic deconvolution [6]. This has the added benefit of lessening linear time-invariant channel effects [6]. Using homomorphic deconvolution is based on the premise that the vocal tract can be modeled as a linear time invariant filter [10].

2.2.2 Creating MFCCs. The feature vectors used in this thesis are MFCCs computed for windowed segments of each utterance. This multi-step process is begun by fixing window and step sizes. The window size determines the duration of a segment of the utterance to consider. The step size indicates how far to shift the window along the duration of an utterance from the beginning of the previous window. The calculation of an MFCC vector begins by taking a 20 ms window of an utterance [12] and determining whether it contains voiced speech or not. If the window contains voiced speech, preemphasis is performed using the common preemphasis coefficient of 0.97 [18]. Next, the magnitude of the discrete Fourier transform (DFT) is calculated, and a triangular filterbank is placed across the spectrum. The filters are placed such that the beginning of the next filter is at the center frequency of the previous filter. Figure 2.1 illustrates how the filters divide the frequency spectrum. The log of the magnitude of the triangular filter outputs are

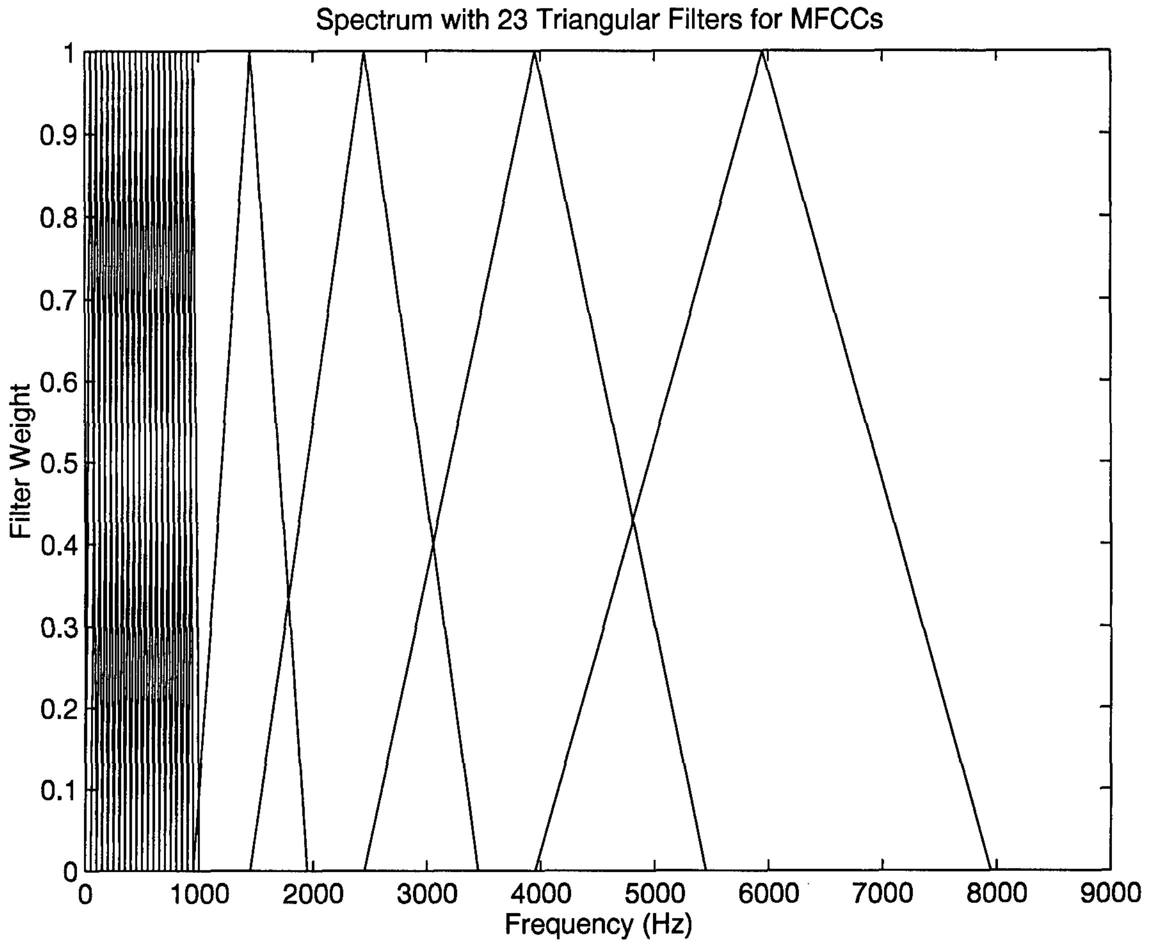

Figure 2.1 Filterbank for MFCCs

then calculated and a discrete cosine transform performed with the resulting coefficients forming the MFCC vector for the given time window [5]. Next, the time window is shifted along the utterance's duration by the step size, a common value is 10 ms [12]. The entire process is then repeated to until a window contains the end of the utterance.

While the concepts for generating MFCCs are common among references, there is some variation in the actual calculations [5,11,18]. For this thesis, the method of [18] was used. Using this method, MFCCs are calculated as

$$c_k = \sqrt{\frac{2}{N}} \sum_{j=1}^{N} m_j \cos\left[\frac{\pi k}{N}(j - 0.5)\right] \qquad \text{for k} = 1 \ldots \text{K} . \tag{2.1}$$

In equation 2.1, N is the number of triangular filters, K is the total number of coefficients desired, and m_j is the log filterbank amplitude for the j^{th} filter from the filterbank along the mel scale. The mel scale is based on experiments on human hearing that suggest filters spaced approximately linearly below 1000 Hz and logarithmically above 1000 Hz [10]. The mel scale may be defined as [18]

$$Mel(f) = 2595 \log \left(1 + \frac{f_{(Hz)}}{700}\right). \tag{2.2}$$

2.3 Gaussian Mixture Models (GMMs)

Once the MFCCs have been calculated, they are used as feature vectors for classification in determining speaker identification and speaker verification.

2.3.1 Theory of GMMs. A GMM is a parametric model consisting of a weighted sum of component Gaussian densities. The model, λ_s, for a given speaker s is defined as a function of the parameters P_i, $\vec{\mu_i}$, and Σ_i such that

$$\lambda_s = \{P_i, \vec{\mu_i}, \Sigma_i\} \qquad \text{for i} = 1 \ldots \text{M} . \tag{2.3}$$

The density of a D-dimensional feature vector, $\vec{x}$, from a sampled window from a given speaker s can be described by

$$p(\vec{x} \mid \lambda_s) = \sum_{i=1}^{M} P_i b_i(\vec{x}), \tag{2.4}$$

where M is the number of mixtures and P_i is the probability or weight of component i such that $\sum_{i=1}^{M} P_i = 1$. The second term, $b_i(\cdot)$, is the density of the Gaussian component i and is found by

$$b_i(\vec{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu_i})'\Sigma_i^{-1}(\vec{x} - \vec{\mu_i})\right], \tag{2.5}$$

where Σ_i is the covariance matrix (assumed to be diagonal [14]) and $\vec{\mu_i}$ is the mean vector for component i. If N is the number of samples in an utterance, the likelihood an utterance came from a given GMM can be found from [4]

$$\mathcal{L}(\vec{U} \mid \lambda_s) = \prod_{n=1}^{N} p(\vec{x}_n \mid \lambda_s), \tag{2.6}$$

which leads to the log-likelihood

$$\log \mathcal{L}(\vec{U} \mid \lambda_s) = \sum_{n=1}^{N} \log p(\vec{x}_n \mid \lambda_s). \tag{2.7}$$

Using Equation 2.4 and Equation 2.7, the log-likelihood of an utterance given a specific GMM is given as

$$\log \mathcal{L}(\vec{U} \mid \lambda_s) = \frac{1}{N} \sum_{n=1}^{N} \log \sum_{i=1}^{M} P_i b_i(\vec{x}_n). \tag{2.8}$$

The first term on the right hand side of Equation 2.8 is used to normalize the likelihood scores so that the scores are independent of the number of voiced segments in a given utterance.

2.3.2 Training GMMs. In order to train the models, a prototype model is created that has the desired number of means and variances for a single component. Initially, the vectors of means are set to zero and the variance vectors are set to one. Next, a speaker is selected, and the speaker's utterances are converted into the previously defined MFCC feature vectors. These feature vectors are then used to estimate by expectation maximization (EM or Baum-Welch Reestimation) [4] the means and variances. The model is "grown" to the desired number of component densities using a binary splitting algorithm. This algorithm takes the component with the greatest weight P_i and creates two new components of half the original weight. The means are the result of adding or subtracting one standard deviation [18]. Additional iterations of the EM algorithm are then performed. The feature vectors for a given speaker's training utterances are repeatedly presented to the speaker's GMM until the desired number of components, 32 [11], is reached.

2.4 Speaker Identification

After a separate GMM has been trained for each speaker in the system, attempts at speaker identification (ID) may begin. For speaker ID, a speaker is prompted to utter some phrase which is then recorded. After determining voiced and unvoiced portions of the utterance, MFCCs are generated from the voiced portions of the utterance. The resulting MFCC feature vectors are then presented to the GMMs for each speaker currently registered in the system. The model with the highest score for a given utterance is believed to belong to the person who gave the utterance. Mathematically this can be represented as selecting speaker r from the set S of all possible speakers for a given utterance $\vec{U}$ such that

$$r = \arg\max_{i \in S} \left\{ \log \left[\mathcal{L}(\vec{U} | \lambda_i) \right] \right\}. \tag{2.9}$$

2.5 Speaker Verification

2.5.1 Introduction. While speaker ID is generally a closed set problem (only speakers registered in the system are allowed to test), applications for speaker verification may be open set problems (speakers not registered in the system may test). For open set problems, it is not enough to know which registered speaker's model scored the highest. The system must determine whether the speaker is really who he or she claims to be (the claimed speaker is referred to as the *claimant*) by comparing the verification score to some absolute threshold.

2.5.2 Cohort Selection. In order to determine whether the speaker is the claimant or not, an additional preprocessing step must be made after all of the GMMs are created. This additional step is the selection of speaker cohorts. The use of cohorts is necessary to normalize the log probabilities to minimize the effects of stress and the natural variability in any given speaker's utterances [7]. Cohorts are chosen from among all of the registered speakers as those that appear most like, called *close cohorts*, and least like, called *far cohorts*, the claimant. To select the cohorts, one utterance from the speaker and one utterance from each of the other registered speakers is required.

Cohorts may be selected from speakers that are more similar or least similar to a given speaker based on a symmetric distortion score [14]. This score is determined by

$$d_{sym}(\lambda_i, \lambda_j) = \log \frac{\mathcal{L}(\vec{U}_i|\lambda_i)}{\mathcal{L}(\vec{U}_i|\lambda_j)} + \log \frac{\mathcal{L}(\vec{U}_j|\lambda_j)}{\mathcal{L}(\vec{U}_j|\lambda_i)}, \tag{2.10}$$

where $\vec{U}_x$ is an utterance from speaker x and λ_x is the GMM for speaker x. A distortion score is determined for a given speaker and all the remaining registered speakers. The resulting scores are then sorted in ascending order. An equal number of maximally spread close cohorts and maximally spread far cohorts are then chosen as reference speakers for each speaker according to [11]. Detailed steps for determining cohorts can be found in Appendix A.

2.5.3 Verification. Verifying a given speaker is the claimant is a multi-step process. First, the speaker is prompted to say some phrase and the utterance, $\vec{U}$, is recorded. Next, any desired filtering, such as voiced/unvoiced determination and preemphasis, is performed on the utterance. Third, the utterance is converted to MFCCs. The resulting MFCC feature vectors are then presented to the claimant's GMM and the likelihood that the utterance was made by the claimant calculated. The MFCCs are then presented to the GMMs for each of the claimant's cohorts. A verification score is obtained according to

$$v(\vec{U}) = \log\left[\mathcal{L}(\vec{U}|\lambda_c)\right] - \frac{1}{B}\sum_{s\in\Omega} \log\left[\mathcal{L}(\vec{U}|\lambda_s)\right], \tag{2.11}$$

where $\vec{U}$ is the utterance, Ω is the claimant's cohort set, B is the the number of cohorts, λ_x is the appropriate GMM for speaker x, and $x \in \{\{c\} \cup \{s \mid s \in \Omega\}\}$. If $v(\vec{U})$ is greater than or equal to some predetermined threshold, the speaker's utterance is accepted as having come from the claimant. Otherwise, the speaker is rejected as the claimant.

2.6 Channel Compensation Techniques

2.6.1 Cepstral Mean Subtraction. Cepstral mean subtraction (CMS) has been found to be helpful in reducing the effects caused by different channel conditions such as different types of microphones or actual phone channels [16]. Performing CMS on a set of MFCC feature vectors requires determining the mean of the set of feature vectors and subtracting the mean from the individual feature vectors in the set prior to determining a likelihood score. The result is normalized feature vectors that lessen the effect of a given channel.

2.6.2 Modified Wiener Filtering. A non-causal Wiener filter, assuming a zero mean signal and noise, can be represented as [8]

$$H(\omega) = \frac{P_s(\omega)}{P_s(\omega) + P_n(\omega)}, \tag{2.12}$$

where the power spectral densities of the desired signal, $P_s(\omega)$, and the noise signal, $P_n(\omega)$, are known *a priori*. For this thesis, however, $P_s(\omega)$ and $P_n(\omega)$ are not known *a priori* and may vary throughout the duration of utterance. An initial approximation of the expected $P_s(\omega)$ is made so that the modified Wiener filter may be modeled as

$$H_{mw}(\omega) = \frac{P_c(\omega)}{P_s(\omega) + P_n(\omega)}, \tag{2.13}$$

where $P_c(\omega)$ is determined by averaging the DFT from window size segments of each of the claimant's training utterances. Since it is impossible to separately determine $P_s(\omega)$ and $P_n(\omega)$ in a real-world scenario, the sum of these terms may be approximated by the measured power spectral density $P_u(\omega)$ so that the modified Wiener filter may be approximated as

$$H_{mw}(\omega) \approx \frac{P_c(\omega)}{P_u(\omega)}, \tag{2.14}$$

where there are no cross terms if the desired signal and the noise are statistically independent. By manipulating the terms, Equation (2.14) becomes

$$H_{mw}(\omega) = \frac{1}{\frac{P_u(\omega)}{P_c(\omega)}} \tag{2.15}$$

and the filter becomes a function of a claimant's average spectrum and the spectrum of the utterance being considered.

The impulse response, $h_{mw}(n)$, is calculated from the inverse DFT of $H_{mw}(\omega)$ and convolved with the original utterance to obtain a new utterance $\hat{U}$. The new utterance is then broken into voiced and unvoiced segments, and the MFCCs are calculated for the voiced segments. These feature vectors are then presented to the appropriate GMMs for identification or verification.

2.7 Summary

This chapter presented the theoretical background for the speaker ID and verification problems. First, the generation of MFCC feature vectors was described. Next, a mathematical basis for GMMs was given. A brief outline was then given on how MFCCs and GMMs are used to classify utterances for determining or verifying a speaker's identity. Finally, two techniques for channel compensation, CMSBL and modified Wiener filtering, were presented.

III. Approach and Methods

3.1 Introduction

In this chapter, brief explanations of the equipment, software, and speaker databases used are provided. Additionally, the procedures used for determining a baseline for speaker identification and speaker verification are given. The steps for performing the modified Wiener filter approach are laid out along with the steps taken in MFCC generation and the training and testing of the GMMs in order to perform identification and verification.

3.2 Computer Equipment and Tools

Before beginning any undertaking, it is important to have the correct tools. The experiments conducted for this thesis were performed on Ultra 1 computers by Sun Microsystems. One of the machines had a CD-ROM drive capable of reading the TIMIT and NTIMIT databases which were transferred to an external 4.0 GB disk drive. All of the Ultras ran Sun Microsystems' Solaris 2.5 operating system.

Working with the data from either corpus was done through one of three tools. The operating environment was set up for UNIX C shells. Shell programs were used to create directory structures, lists of speakers and utterances, and to automate calls to other tools. Mathworks' *MATLAB version 4.2c* as well as Entropic's *HTK - Hidden Markov Toolkit version 2.0* and *ESPS version 5.1* were used to manipulate utterances, generate MFCCs, train Gaussian mixture models (GMMs), and obtain log-likelihood scores.

3.3 Speaker Databases

Developed by Texas Instruments (TI), Inc. and the Massachusetts Institute of Technology (MIT), the TIMIT database was initially designed for speech recognition. It was created under nearly ideal conditions using the same recording equipment over a short period of time with 630 speakers. The recordings were made with an 8 kHz bandwidth at a sampling rate of 16 kHz [1]. This means that there is a low degree of inter-session variability in a given speaker's utterances, acoustical noise, and equipment variability.

The NTIMIT database was created by NYNEX using the original TIMIT database [2]. The original utterances were played back through an artificial mouth into a carbon-button telephone handset. The resulting speech was then transmitted to central offices of local or long-distance systems and looped back for recording at a 16 kHz sampling rate.

The databases are originally broken into test and training sections with no common speakers between the sections. These sections are further subdivided into eight dialect regions based on a speaker's primary residence during their lifetime. The dialect regions are then divided into speakers with each speaker's 10 utterances in the speaker's subdirectory.

3.4 Baseline

Initial experiments were done to reproduce Reynolds' previous methods [12] and results [13,14] for identification and verification using the TIMIT database. The process for

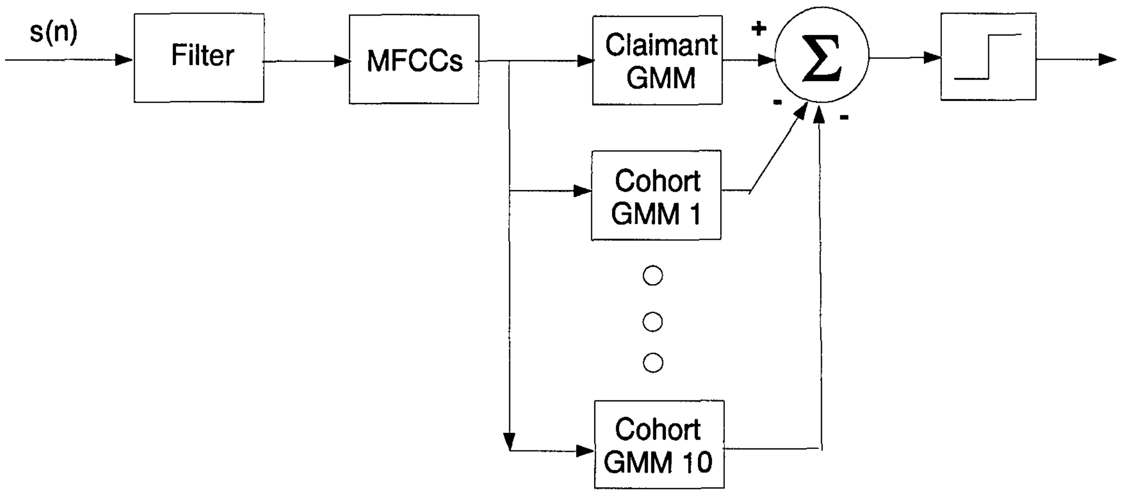

Figure 3.1 Baseline Speaker Verification Process

reproducing the verification results is illustrated by Figure 3.1. For the baseline, filtering was dependent on the method under consideration. Additional details are provided in the following sections.

3.4.1 Training and Testing. Each speaker has 10 utterances divided into SA1 and SA2 (both of which are common to all speakers), as well as three SI sentences and five SX sentences. For the reproduction of results, training for all speakers was done using both SAs, all SIs, and the first three (in UNIX *ls* order) SX utterances from the TIMIT database. All testing was done on the last two SX utterances. Initially, training and testing were only done on the 168 speakers from across all eight dialect regions in the test section for all three methods under consideration. In an additional round of experiments, all 630 speakers were used only for the straight TIMIT training and testing cases as well as the straight NTIMIT testing cases for speaker ID and verification.

3.4.2 Straight TIMIT & NTIMIT. When utterances were taken straight from the original database recordings with no warping or filtering, they are referred to as "straight" utterances. Due to the formatting of the original databases, these utterances were manipulated by removing the header information using the ESPS *bhd* command and then using the UNIX utility *dd* to swap an utterance's byte order. The m-file detvoice.m (see Appendix B) was used to determine the voiced and unvoiced segments based on the hand-labeled phonetic transcriptions provided with each utterance in the TIMIT and NTIMIT databases. The transcriptions were used to facilitate reproduction of results. MFCCs were created using HTK's *HCopy* and the HTK configuration file *hconfig*. The hconfig file was set to use 24 filters in the filterbank and return 23 coefficients. GMMs were created from each speaker's eight training utterances while likelihood scores were generated from the two test utterances from each speaker.

3.4.3 CMSBL. To reproduce the cepstral mean subtraction with bandlimiting (CMSBL) method a separate HTK hconfig file was made. This time, however, the HTK hconfig file was also set to bandlimit the utterances from 400 - 3200 Hz in accordance with [12] before calculating the MFCCs. The voiced and unvoiced detection was again performed using detvoice.m to access the hand-labeled phonetic transcription files. New GMMs were created for each speaker from these MFCCs and new log-likelihood scores were computed for each speaker's test utterances.

3.5 Modified Wiener Filtering

The process of manipulating utterances using the modified Wiener filtering approach was similar to the straight and CMSBL methods. The difference in the preprocessing is

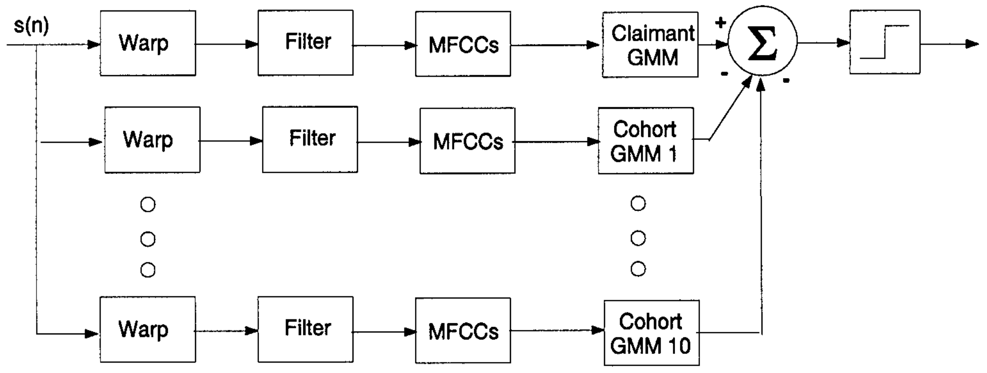

Figure 3.2 Modified Wiener Filter Speaker Verification Process

illustrated in Figure 3.2. As Figure 3.2 illustrates, modified Wiener filtering was applied to each of the utterances prior to generating the MFCCs. Each utterance was filtered using the claimant's long-term average spectrum. This was done by approximating the frequency response of the channel using

$$H_{mw}(\omega) \approx \frac{P_c(\omega)}{P_s(\omega) + P_n(\omega)} = \frac{P_c(\omega)}{P_u(\omega)}, \tag{3.1}$$

where $P_c(\omega)$ was determined by averaging the DFT from window size segments of each of a claimant's training utterances, $P_s(\omega)$ and $P_n(\omega)$ were the desired signal and the noise signal, respectively, and P_u was the measured power spectral density.

The impulse response of the filter, $h_{mw}(n)$, was calculated and convolved with the original utterance to obtain a new utterance $\hat{U}$. The new utterance was then broken into voiced and unvoiced segments using the appropriate hand-labeled phonetic transcription files and the MFCCs calculated for the voiced segments. New GMMs were generated for this method also. The process was then repeated for each speaker's two test utterances using all possible speakers as the claimant.

The inspiration for this approach came from work done with Wiener filtering of distorted images [8]. By using this modified approach on speech corrupted by a channel, the signal resulting from the warping should skew the spectrum of an utterance to be more like those of the claimant and the claimant's cohorts as seen earlier in this chapter. Additionally, this approach will smear the spectra, just as its image counterpart, which may result in greater speaker ID errors but hopefully fewer verification errors similar to the CMSBL method.

3.6 MFCC Calculation

Once header information had been removed, the byte order of the utterances swapped, and the voiced and unvoiced segments determined, the MFCCs were calculated using HTK's *HCopy* command. This command was invoked either manually or by the m-file modwmfcc.m (see Appendix B), to compute the MFCCs for voiced segments according to

$$c_k = \sqrt{\frac{2}{N}} \sum_{j=1}^{N} m_j \cos\left[\frac{\pi k}{N}(j-0.5)\right], \qquad \text{for k} = 1 \ldots \text{K}, \tag{3.2}$$

where K was the number of desired coefficients, N was the number of filters to use in the filterbank, and m_j was the log amplitude of the j^{th} filter of the triangular filters along the mel-scale defined by

$$Mel(f) = 2595 \log\left(1 + \frac{f_{(Hz)}}{700}\right). \tag{3.3}$$

The hconfig file used with *HCopy* varied depending on whether straight or CMSBL MFCCs were desired.

3.7 GMM Generation

3.7.1 HMM or GMM? Once the MFCCs had been calculated, models were developed for each speaker under each of the three methods. For expediency and the ability to facilitate a reproduction of the experiments, HTK was used to develop a degenerate form of hidden Markov model (HMM). HTK creates HMMs that always have initial and ending non-emitting states. By including only one state between them and "growing" that state into multiple Gaussian mixtures, a Gaussian mixture model (GMM) may be created. From this point on, the HTK HMMs will simply be referred to as GMMs.

3.7.2 Prototype GMM Development. A prototype model was first created according to the specifications in *The HTK Book* [18]. A sample prototype for straight data is included below.

```
 1. <BeginHMM>
 2.     <NumStates> 3 <VecSize> 23 <MFCC> <nullD> <diagC>
 3.     <StreamInfo> 1 23
 4.     <State> 2 <NumMixes> 1
 5.     <Stream> 1
 6.     <Mixture> 1 1.0
 7.     <Mean> 23
 8.         0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
 9.     <Variance> 23
10.         1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
11. <TransP> 3
12.     0.000e+0 1.000e+0 0.000e+0
13.     0.000e+0 6.000e-1 4.000e-1
14.     0.000e+0 0.000e+0 0.000e+0
15. <EndHMM>
```

All prototypes and model definitions begin with the line "<BeginHMM>". The second line of the prototype indicates the total number of states that the model should have, the size of the vectors it should expect, the type of data, any special handling, and the type of covariance matrix to assume. In this thesis, only three states were required (the initial non-emitting state, the state of interest, and the final non-emitting state). For consistency with Reynolds, the vector size was chosen to calculate 23 MFCCs using a bank of 24 filters [11]. The third line indicates the number of sources that will be presented to the model and their size. Line 4 determines the state number and its number of mixtures. Line 6 indicates the mixture number for a given state and the probability of that mixture.

For each mixture in a state, other than the initial and final states, the state number and the desired number of means and variances for the state are also indicated, e.g., lines 7 and 9. Below the number of means and variances, an appropriate number of constants are entered as place holders in lines 8 and 10. The actual values are not important, only that there be as many as indicated by VecSize. To speed implementation, the means are typically zero and the variances one. Line 11 indicates the beginning of the transition probability matrix for the HMM and indicates the number of states. In practice, the transition probabilities should be close to the final results, but do not need to be exact.

It should be noted that failure to split the model from one-to-two and then two-to-four components before reestimating had suboptimal results. Failure to make the two-to-four split before reestimating resulted in returning to a single component after reestimation following the initial split rather than adjusting to two components.

3.7.3 Training the GMM. Once the MFCCs were generated for all the utterances and a prototype GMM developed, an individual speaker's GMM was "grown" to the desired size of 32 components [11, 12]. This was done by selecting a given speaker and the associated training MFCCs and presenting them to the GMM. This process was done using the C-shell script gmm2maker (found in Appendix B) and is illustrated with the following pseudo code.

1. Initialize the model using the speaker's training utterances and *HInit*
2. Reestimate the model using all training utterances with *HRest*
3. Perform a binary split of the mixture using *HHEdit* with *MU 2* (MU q is an HTK command that increases the number of components in the mixture to q)
4. Perform a binary split of both mixtures using *HHEdit* with *MU 4*
5. Reestimate the model with *HRest* using all training utterances
6. Perform a binary split of the top two mixtures using *HHEdit* with *MU* $(2 \times x)$ where $x = 3 \ldots M/2$
7. Reestimate the model using all training utterances with *HRest*
8. Repeat 6 and 7 until the desired number of mixtures is reached

3.7.4 Probability Scores. With GMMs created for all of the speakers, log-likelihood scores for a a given speaker's utterance were obtained using the C-shell script uttscores2.c (in Appendix B). The scores were calculated based on Equation 3.4 and

Equations 2.4 and 2.5 as

$$\log \mathcal{L}(\vec{U} \mid \lambda_s) = \frac{1}{N} \sum_{n=1}^{N} \log\left[p(\vec{x}_n \mid \lambda_s)\right], \tag{3.4}$$

where N is the number of voiced samples in an utterance.

The scores were calculated using HTK's *HVite* command. *HVite* presented each utterance's MFCCs separately to each of the 168 or 630 models and saved the scores. Varying the appropriate parameters in uttscores2.c obtained the results for all 630 speakers. The script was used to generate scores for both SA utterances as well as the last two SX utterances from each speaker. The scores from the SA utterances were used to determine initial thresholds for verification results with a minimal equal error rate. The scores from the SX utterances were used to determine the system's success with the SA thresholds as well as to detemine an equal error rate of their own.

3.8 Speaker Identification

Speaker identification required using MATLAB to determine which GMM generated the highest log-likelihood score for a given utterance by reading in the files created by uttscores2.c (see Appendix B). If the index of the highest score corresponded to the index of the utterance's speaker, the trial was considered to have correctly identified the speaker. Otherwise, the trial was considered to have made an error in speaker identification. Winning models were chosen based according to

$$r = \arg\max_{i \in S} \left\{ \log\left[\mathcal{L}(\vec{U} | \lambda_i)\right] \right\}. \tag{3.5}$$

This process was repeated for all speakers and their corresponding SA utterances and separately for SX testing utterances using the m-file spkrid.m (see Appendix B) with the results from each iteration saved to separate files.

3.9 *Speaker Verification*

3.9.1 Cohort Selection. Having already calculated the probability scores for each speaker's SA1 and SA2 utterances for all GMMs under consideration, determining the 10 cohorts was straight-forward. (Note: this differs slightly from [11] in which cohorts were determined by obtaining a probability score from the concatenation of all the training MFCCs.) First, a speaker i and a different speaker j were chosen and a distortion score was determined according to Equation 3.6.

$$d_{sym}(\lambda_i, \lambda_j) = \log \frac{\mathcal{L}(\vec{U}_i|\lambda_i)}{\mathcal{L}(\vec{U}_i|\lambda_j)} + \log \frac{\mathcal{L}(\vec{U}_j|\lambda_j)}{\mathcal{L}(\vec{U}_j|\lambda_i)}, \qquad for \quad i \neq j, \tag{3.6}$$

where $\vec{U}_i$ is speaker i's SA1 utterance, λ_x is speaker x's GMM, and $\vec{U}_j$ is speaker j's SA2 utterance.

These scores were then sorted in ascending order. The five maximally spread close cohorts were the distortion scores closest to zero and, therefore, believed to sound most like the speaker but were not duplicates (speakers whose scores were not necessarily adjacent to one another). The five maximally spread far cohorts were those whose distortion scores were furthest from zero and, therefore, were believed to sound least like the speaker but who were not duplicates (speakers whose distortion scores were not necessarily adjacent to one another). A detailed algorithm for cohort selection can be found in Appendix A. The process was automated to find cohorts for all speakers using the m-file maincohort.m (see Appendix B). Separate cohort sets were found for the straight, CMSBL, and modified Wiener filter methods.

3.9.2 Verification Scores. After cohorts were determined for each speaker, verification scores were calculated for each speaker's test SX utterance where each of the registered speakers posed as the claimant. Verification scores were calculated according to

$$v(\vec{U}) = \log\left[\mathcal{L}(\vec{U}|\lambda_c)\right] - \frac{1}{B}\sum_{s\in\Omega} \log\left[\mathcal{L}(\vec{U}|\lambda_s)\right], \tag{3.7}$$

where $\vec{U}$ is the utterance, Ω is the claimant's cohort set, B is the number of cohorts in the set, λ_x is the appropriate GMM for speaker x, and $x \in \{\{c\} \cup \{s \mid s \in \Omega\}\}$. While verification scores were calculated all cases, only the results for testing without cohorts were reported for consistency with [11].

3.10 Equal Error Rate

Before determining the equal error rate (EER), the verification scores for all of the tested SX utterances for each of the speakers were sorted in ascending order. An initial arbitrary threshold was chosen and the number of false accepts and false rejects determined. The rates for false accepts and false rejects were calculated and the EER assigned as the average of the false accept and false reject rates. The final EER was determined by finding the threshold for which the difference between false accept and false reject rates was minimum.

3.11 Summary

This chapter has presented the approach and methods used in this thesis. A brief listing of the computer hardware, software, and databases used was provided. Discussions on how MFCCs were generated and on the procedures used in building and training a GMM were provided. Additionally, the methods used for speaker identification and speaker verification, including the selection of cohorts, were also given. Finally, the method used for calculating the EER was discussed.

IV. Experimental Results and Analysis

4.1 Introduction

The results from the various trials are presented in terms of process, the corpus, and the method used. The first section covers the speaker ID results for both databases and each method. Then the speaker verification results are presented. Lastly, the first known speaker ID and verification results for the entire 630 speakers in straight TIMIT and NTIMIT are presented. For each process, training was done only on TIMIT utterances.

4.2 Identification

4.2.1 TIMIT. Tables 4.1, 4.2, and 4.3 give the results from testing against the 168 test speakers of TIMIT using the SA and SX2 utterances from TIMIT. Of the three methods (straight, CMSBL, and modified Wiener filtering), the straight method had the best speaker ID results for the test SX utterances with 100% accuracy for the 168 speakers. The CMSBL approach was a close second making only three errors on the test utterances for 99.1% correct identification. The modified Wiener approach performed the worst with 84.2% correct identification. The decrease in correct identification by the CMSBL approach supports Reynolds' assertion that cepstral mean subtraction will reduce performance when training and testing conditions are the same [12].

Table 4.1 Straight TIMIT Identification for 168 speakers

Utterance	# Errors	# Correct	Errors %	Correct %
SA1 and SA2	0	336	0	100
test SX	0	336	0	100

Table 4.2 CMSBL TIMIT Identification for 168 speakers

Utterance	# Errors	# Correct	Errors %	Correct %
SA1 and SA2	0	336	0	100
test SX	3	333	0.893	99.1

4.2.2 NTIMIT. Table 4.4 illustrates the results from testing the NTIMIT SX2 utterances against the 168 test speakers. Again, the GMMs were trained on TIMIT utterances and tested against the NTIMIT ones, and providing tests done under channel

Table 4.3 Modified Wiener filter TIMIT Identification for 168 speakers

Utterance	# Errors	# Correct	Errors %	Correct %
SA1 and SA2	47	289	14.0	86.0
test SX	53	283	15.8	84.2

mismatch conditions. This time, the CMSBL method performed the best of the three methods with 27.4%. The modified Wiener filter method was a distant second with only 4.17% correct and the straight method was third with 3.57%.

Table 4.4 Straight NTIMIT Identification for 168 speakers

Utterance	# Errors	# Correct	Errors %	Correct %
Straight	324	12	96.4	3.57
CMSBL	244	92	72.6	27.4
Mod W	322	14	95.8	4.17

4.3 Verification

4.3.1 TIMIT. Tables 4.5, 4.6, and 4.7 illustrate the results from testing the TIMIT SX2 utterances against the 168 TIMIT test speakers. The verification EER for the TIMIT test utterances was extremely close (within one false reject) to Reynolds' $\approx$ 0.24% EER [14]. This discrepancy is likely the result of the subtle difference in cohort selection. The CMSBL method produced the best results with the straight method a close second and the modified Wiener method a distant third. In Tables 4.5, 4.6, and 4.7, a "*" indicates misleading EERs due to averaging.

Table 4.5 Straight TIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
SA1 and SA2	10.375	0.576	148	2	0.557	0.595
test SX	10.375	0.504 *	375	1	0.711	0.298
test SX	10.650	0.595	314	2	0.595	0.595

In these verification tables, the first test SX threshold is taken from the threshold obtained for training SA threshold scores as would be done in a real-world system. The second test SX threshold was determined by allowing the system to find the EER and threshold for the test SX utterances. While this second test SX threshold could not be done in a practical system, the result provides a means of gauging how well the system might have done with either additional training utterances to determine the threshold or

Table 4.6 CMSBL TIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
SA1 and SA2	6.527	0.338	200	1	0.379	0.298
test SX	6.527	0.147*	155	0	0.294	0.000
test SX	6.837	0.262	119	1	0.226	0.298

Table 4.7 Modified Wiener filter TIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
SA1 and SA2	4.5081	7.85	4,202	26	7.97	7.74
test SX	4.5081	8.77	3,758	35	7.12	10.4
test SX	4.2172	8.11	4,313	27	8.18	8.04

in a text-dependent scheme. For the straight and CMSBL methods, the "better" EERs increased the number of false rejects by one, while the number of false accepts was reduced. For the modified Wiener filter method, the new ERR has similar effects but with more drastic results. The number of false rejects decreased form 35 to 27 while the number of false accepts rose from 3,758 to 4,313.

4.3.2 NTIMIT. Tables 4.8, 4.9, and 4.10 give the results of testing the NTIMIT SX2 utterances against the 168 test speakers of TIMIT under mismatched testing and training conditions. The results were similar to those for TIMIT. Again, the CMSBL method produced the best results with an EER of 19.7%, almost half that of either the straight method's 37.2% or the modified Wiener filter's 39.0%. As in the previous tables, a "*" indicates misleading EERs due to averaging and "-" indicates methods that are not practical since most real users are kept out. For the NTIMIT verification tables,

Table 4.8 Straight NTIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
test SX	10.375-	49.7*	21	334	0.0398	99.4
test SX	10.650-	49.7	18	334	0.0341	99.4
test SX	1.19	37.1	19,686	124	37.3	36.9

the first test SX threshold is taken from the threshold obtained for training TIMIT SA threshold scores as would be done in a real-world system. The second test SX threshold was determined from the threshold for the lowest EER from the TIMIT test SX utterances.

Table 4.9 CMSBL NTIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
test SX	6.527 -	49.1 *	13	330	0.0246	98.2
test SX	6.837 -	49.7	7	334	0.0133	99.4
test SX	1.237	19.6	10,359	66	19.6	19.6

Table 4.10 Modified Wiener filter NTIMIT Verification for 168 speakers

Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
test SX	4.5081 -	46.8 *	528	311	1.00	92.6
test SX	4.2172 -	46.2 *	751	306	1.42	91.1
test SX	0.6549	39.0	20,566	131	39.0	39.0

The final test SX threshold was determined by allowing the system to find a more accurate EER and threshold for the NTIMIT test SX utterances.

4.4 Entire Corpus

Tables 4.11 and 4.12 give the results for speaker identification and speaker verification, respectively, of testing all 630 speakers of TIMIT using utterances from TIMIT and NTIMIT. Table 4.11 illustrates that, even for the entire 630 speaker set, the straight method yields excellent results for matched training and testing conditions. When testing with NTIMIT utterances, however, the result is almost the inverse with only 10 correct identifications. These results emphasize the need for additional processing of the utterances prior to identification under channel mismatch conditions.

Table 4.11 Straight Identification for 630 speakers

Corpus	Utterance	# Errors	# Correct	Errors %	Correct %
TIMIT	SA1 and SA2	0	1260	0.00	100
	test SX	1	1259	0.0794	99.9
NTIMIT	test SX	1250	10	99.2	0.794

Large population speaker verification results are given in Table 4.12. For the TIMIT SX testing utterances, the first threshold (10.8959) was determined from the SA utterances. The second threshold (10.8682) yielded the best EER for the TIMIT SX testing utterances. For the NTIMIT SX testing utterances, the three thresholds represent the TIMIT SA, the TIMIT testing SX, and NTIMIT testing SX error rates.

Table 4.12 Straight Verification for 630 speakers

Corpus	Utterance	Threshold	EER %	#fa	#fr	FA %	FR %
TIMIT	SA1 and SA2	10.8959	0.510	4,242	6	0.544	0.476
	test SX	10.8959	0.666	4,813	9	0.617	0.714
		10.8682	0.596	4,885	7	0.626	0.556
NTIMIT	test SX	10.8959 -	49.8 *	439	1,255	0.0563	99.6
	test SX	10.8682 -	49.8 *	449	1,255	0.0576	99.6
	test SX	1.267	38.7	302,071	488	38.7	38.7

Despite increasing the number of speakers used by Reynolds from 168 to 630, the EER for the TIMIT SX case changed by 0.16% for the SA threshold and only 0.04% for the second test SX threshold. The difference in thresholds across mismatched channel conditions in Table 4.12 illustrates just how difficult the problem was. Mismatched conditions not only influenced the error rates for a given threshold but dramatically affected the EER threshold as well.

4.5 Summary

In this chapter, Reynolds' results for speaker identification were reproduced for straight TIMIT. His results for speaker verification using only the 168 test speakers from TIMIT were also reproduced to within one false rejection. The CMSBL method performed almost as well as the straight method when applied to TIMIT utterances and better than the straight method when applied to NTIMIT utterances. The modified Wiener filtering method was found to be significantly less effective for speaker identification of TIMIT utterances and only slightly better than the straight method for NTIMIT utterances. The modified Wiener filtering method performed significantly worse than CMSBL for both TIMIT and NTIMIT testing conditions. For mismatched channel conditions, the results for speaker verification were similar to speaker identification with CMSBL doing the best. The straight method did an excellent job of speaker ID for the large (630) speaker populations when testing on TIMIT utterances with 99.9% correct. However, the results for NTIMIT, only 0.794% correct, indicate the need for some form of preprocessing under mismatched channel conditions.

V. Conclusions

5.1 Conclusions

This thesis reproduced previously obtained results for speaker identification and verification using MFCCs and GMMs for a population size of 168 speakers. The previous work was then extended to a larger population of 630 speakers. A modified Wiener filtering approach was used in an attempt to minimize channel mismatch effects on speaker verification. It was found, however, that this approach yielded worse performance by a factor of two than traditional cepstral mean subtraction with bandlimiting. The modified Wiener approach was 2% worse than using straight models with unfiltered speech. Three conclusions can be drawn from these results. First, the results for all 630 speakers for matched training and testing conditions imply that there is ample room in the 23 MFCC feature space for speaker ID and verification. The channel mismatch conditions, however, require some sort of preprocessing prior to speaker ID or verification. Second, while the modified Wiener filtering method looked promising from a mathematical standpoint, no filtering of an utterance yields better results at a lower computational cost. Lastly, CMSBL yields the best verification results for matched or mismatched conditions with only a minor degradation in speaker ID under matched conditions.

5.2 Future Study

Suggestions for future research include the following:

- Convolve the inverse DFT of the speaker's average spectra with the original utterance. This method was found serendipitously and the results sounded subjectively better than the original TIMIT utterances. This is along the lines of Stockam's efforts [17].
- Try Avendro and Hermanksy's method of channel normalization [3].
- Create versions of YOHO similar to NTIMIT and CTIMIT to test channel effects on a standard speaker verification corpus instead of a speech recognition corpus.
- Investigate the use of speaker ID results to improve speaker verification results.

- Investigate the use of speaker-dependent thresholds instead of using a global threshold for speaker verification.
- Continue the search for feature vectors that are more robust to channel effects than MFCCs.

Appendix A. Cohorts

A.1 Calculating Symmetric Distortion Scores

Cohorts are chosen from among all of the registered speakers as those that appear most and least like the claimant. This algorithm is based on the one given by Reynolds [14]. To select a speaker's cohorts, one utterance from the speaker and one utterance from each of the other registered speakers are required. Instead of a single utterance from a speaker, a concatenation of all the speaker's training utterances may also be used. A symmetric distortion score is calculated for a given speaker i and one of the other registered speakers j as

$$d_{sym}(\lambda_i, \lambda_j) = \log \frac{p(\vec{U}_i|\lambda_i)}{p(\vec{U}_i|\lambda_j)} + \log \frac{p(\vec{U}_j|\lambda_j)}{p(\vec{U}_j|\lambda_i)}, \qquad for \quad i \neq j, \tag{A.1}$$

where $\vec{U}_x$ and λ_x are utterances by and speaker models for speaker x, respectively. In this way, a distortion score is calculated for the given speaker i and each of the remaining registered speakers. The resulting scores are then sorted in ascending order.

A.2 Selecting Cohorts

Set B to the total number of desired cohort speakers to be selected, where B is even when choosing both close and far cohorts. For selecting both close and far cohorts, set $S = B/2$. For selecting only close or only far cohorts, set $S = B$. To avoid having cohorts that are extremely similar, a maximally spread algorithm is used per Reynolds [11].

A.2.1 Chosing Close Cohorts. For a given speaker i, this process chooses the registered speakers that "sound" most like the given speaker. Choose N_{tot} speakers with the smallest distortion scores to create a pool of potential close cohorts PC_i for speaker i. The potential close cohorts should exclude speaker i, and N_{tot} should be chosen large enough such that $N_{tot} > S$.

Step 0: Move the closest speaker (i.e., the one with the smallest distortion score) from PC_i to C_i, the final set of close cohort speakers for speaker i.

Set $C' = 1$, where C' is the number of cohort speakers already selected for speaker i.

Step 1: Move speaker θ from PC_i to C_i where

$$\theta = \arg\max_{c \in PC_i} \left[\frac{1}{C'} \sum_{b \in C_i} \frac{d(\lambda_b, \lambda_c)}{d(\lambda_i, \lambda_c)} \right] \tag{A.2}$$

Set $C' = C' + 1$

Step 2: Repeat Step 1 until $C' = S$.

A.2.2 Chosing Far Cohorts. For a given speaker i, this process chooses the registered speakers that "sound" least like the given speaker. First, choose the $N_{tot} > S$ speakers, excluding speaker i, who had the largest distortion scores in order to create a pool of potential far cohorts, PF_i.

Step 0: Move the furthest speaker (i.e., the one with the largest distortion score) from PF_i to F_i, the final set of far cohort speakers.

Set $F' = 1$, where F' is the number of far cohort speakers already selected.

Step 1: Move speaker τ from PF_i to F_i where

$$\tau = \arg\max_{f \in F_i} \left[\frac{1}{F'} \sum_{b \in F_i} d(\lambda_b, \lambda_f) \times d(\lambda_i, \lambda_f) \right]. \tag{A.3}$$

Let $F' = F' + 1$.

Step 2: Repeat Step 1 until $F' = S$.

When both close and far cohorts are desired, create a total cohort set, Ω_i, for each speaker i using both sets of cohorts so that

$$\Omega_i = \{C_i \cup F_i\}. \tag{A.4}$$

If only close cohorts are desired, use $\Omega_i = \{C_i\}$. Calculating symmetric distortion scores and selecting cohorts is then repeated for all of the registered speakers.

Appendix B. C-Shell Scripts and MATLAB m-files

B.1 m-files [1]

```
%
% detvoice.m
%
% function vuv=detvoice(curphon,phndb);
%
% Description: This function determines whether the phoneme interval containing
%              the current frame of speech is voiced or unvoiced.
%
% Author: Capt Al Arb, USAF
% Date: 30 Jul 96
% Modified:
%
% Input parameters:
%      curphon: The phoneme label for the current frame of speech.
%       phndb: The TIMIT phoneme data base matrix.
%
% Output Parameter:
%     vuv: voiced/unvoiced.  1=voiced, 0=unvoiced.
%
% Subroutines directly called:
%     none
%
% Subroutines indirectly called:
%     none
%

function vuv=detvoice(curphon,phndb);

vuv=0;
count=0;
done=0;

%
% Loop until we find the label
%

while ~done
  count=count+1;
%
% If the DB entry = curphon, we found it.
%
  if phndb(count,1:4)==curphon
    done=1;
    phn=count;

  %
  % Or if we are at the end of the file, stop and assume unvoiced.
  %

  elseif count==length(phndb(:,1))
    done=1;
    phn=0;
  end;

end;

if phn
  %
```

[1] *For printing purposes the m-files were concatenated into a single file. The line numbering in the right margin is correct for the concatenated file.*

```
  % If the category is VOICED, set vuv to 1.
  %
  if phndb(phn,5:10)=='VOICED'
    vuv=1;
  end;
end;

%
% distmtrx.m
%
% [dscores] = distmtrx(sa1matrix, sa2matrix)
%
% This function calculates the distortion metric scores for a given
% speaker when provided the proper inputs.
%
% Key assumptions: Scores are log-likelihoods
%                  Matrices are square
%                  Rows indicate utterance from a given speaker
%                  Columns indicate model for a given speaker
%
% Input:
%    sa1matrix    matrix of scores for each model for one utterance
%
%    sa2matrix    matrix of scores for each model for second utterance
%
% Output:
%    dscores      a matrix of distortion metric scores
%
% Created by Capt  R. Brian Reid
% Date:  8 Aug 1997
%
% References: Columbi et. al. "Allowing Good Imposters to Test"
%             based on Reynolds
%
% Last modified:  14 Aug 1997
%                 1005
%
function  [dscores] = distmtrx(sa1matrix, sa2matrix)

numspkrs = size(sa1matrix,2);

dscores = zeros( numspkrs );

for iloop = 1 : numspkrs,

  for jloop = 1 : numspkrs,

    dscores( iloop, jloop) =  sa1matrix(iloop, iloop) - sa1matrix(iloop, jloop) + ...
        sa2matrix(jloop, jloop) - sa2matrix(jloop, iloop);

  end;

end;

%
% emailmsg
%
% emailmsg(emailaddress,message);
%
% Used to send email message to user in UNIX OS.
% by: Capt. Edward M. Ochoa, GEO-96D

function emailmsg(emailaddress,message);

UCMD1=sprintf('echo "%s" > /tmp/emailmsg.txt',message);
```

```
UCMD2=sprintf('echo "~s %s" >> /tmp/emailmsg.txt',message);
UCMD3=sprintf('echo "~." >> /tmp/emailmsg.txt');
UCMD4=sprintf('mail %s < /tmp/emailmsg.txt',emailaddress);
UCMD5=sprintf('!rm /tmp/emailmsg.txt');

UCMDS=['! ' UCMD1 ';' UCMD2 ';' UCMD3 ';' UCMD4 ';'];

eval(UCMDS)
eval(UCMD5)

%
% exvu4htk.m
%
% [voicedspeech,unvoiced] = exvu4htk(data,uttfile,fs,wlength,fstep,phnfile);
%
% Inputs:
%     data        actual waveform data (byte swap if necessary)
%     uttfile     file name of utterance to use
%     fs          sample frequency in Hz
%     wlength     window size in seconds
%     fstep       frame step size in seconds
%     phnfile     name of file (*.phn) with phoneme labels
%                 useful if phoneme labels are in a separate directory
%
% Derived from exvoiced4htk.m code by Capt Al Harb
% Modified by  Capt R. Brian Reid
%              19 Jul 1997
%

function [voicedspeech,unvoicedspeech] = exvu4htk(data,uttfile,fs,wlength,fstep,phnfile);

extind=find(uttfile=='.');

if phnfile ==[]
  phnfile = [uttfile(1:extind) 'phn'];
end;

% phnfile

%[phnind,phnval]=readphn(setstr([uttfile(1:extind) 'phn']));
[phnind,phnval]=readphn(setstr([phnfile]));

phndb=loadphndb;

wstart=1;
wend=wlength*fs;
finished=0;
voicedspeech=[];
unvoicedspeech= [];

while ~finished,

    cur_phoneme=find((phnind(:,1)<=wend)&(phnind(:,2)>wend));
    vuv=detvoice(phnval(cur_phoneme,1:4),phndb);

    if vuv,
        voicedspeech=[voicedspeech; data(wstart:wend)];
    else
        unvoicedspeech = [unvoicedspeech; data(wstart:wend)];
    end;
    wstart=wstart+(fstep*fs);
    wend=wstart-1+(wlength*fs);
    if wend > length(data),
        finished=1;
    end;
end;

%
```

```
% fftavgf.m  is a function to calculate the average FFT or just the FFT of
% a segment of a speech file
%
% [avgfft] = fftavgf(data, fftsize, samplefrequency, windowsize)
%
% Inputs:
%   data       Actual data
%   numsamples  Number of samples
%   samplefrequency       sampling frequency data was obtained at
%   window size         time in seconds of the window of speech
%
% Output:
%   avgfft                 Average spectra for the input
%

function [avgfft] = fftavgf(data, fftsize, samplefrequency, windowsize)

% Read in the utterance

% [data, numsamples] = readhtkn(utterance,0);

numsamples = max(size(data));

% Determine the number of samples per chunk of time
timechunks = samplefrequency * windowsize;

% Initialize the average to zero
avgfft = zeros(fftsize,1);
runsum = zeros(fftsize,1);

pointer = 1;

% Incrementally calculate the FFT
numchunks = floor(numsamples / timechunks);

for loop = 1 : numchunks
  temp = abs( fft( data( pointer : pointer - 1 + timechunks ), fftsize ) );
  pointer = pointer + timechunks;

  % Update the runningsum
  runsum = runsum + temp;

end;

% Compute the average as running sum / # of chunks

if numchunks ~= numsamples / timechunks
  temp = abs( fft( data( pointer : length(data) ) ) );

  runsum = runsum + [temp; zeros(fftsize - length(temp), 1) ];

  loop = loop + 1;

end;

avgfft = runsum / loop;


%
% fndcchrt.m
%
% [cohorts] = fndcchrt(distortionmatrix, N, B, spkrnum);
%
% fndcchrt finds the close cohorts for a given distortion matrix.
%
% Inputs:
%    distortionmatrix   a square matrix of distortion scores
%    N           size of pools to use in selecting background speakers
%    B           size of background speakers for maximally spread close
%    spkrnum  indicates number of speaker in distortionmatrix
%
```

```
% Ouput:
%    Bi  vector of pointers to speakers determined to be cohorts
%
% References:
%    Reynolds "Speaker Identification and verification using
%              Gaussian mixture speaker models." Speech Communication
%              17 (1995) p 91-108
%
% Created by Capt R. Brian Reid
%
% Created on 14 Aug 1997
%
% Last Modified: 26 Aug 1997
%                1138
%

function Bi = fndcchrt(matrix, N, B, spkrnum);

Nc = N;
Nf = N;

numspkrs = size(matrix,2);

% Setup
% Find the set of maximally spread close speakers for spkrnum

% Select spkrnum's utterance against all models
Dv = matrix(spkrnum, : );

% Create an index of all of the speakers
indexptr = linspace( 1, numspkrs, numspkrs);

% Remove current speaker from the list
Dv = nixcol(Dv, spkrnum);

indexptr = nixcol(indexptr, spkrnum);

% Sort Dv and get an index back
[Dvnew, index] = sort(Dv);

% Reorganize indexptr according to index
indexptrnew = indexptr(index);

% Select N closest speakers (speakers with smallest distortion)
Ci = indexptrnew(1:Nc);

%
% Step 0: Move the closest speaker from Ci to Bi
%

  Bi = Ci(1);
  Ci = nixcol(Ci, 1 );
  Nc = Nc - 1;
  Bprime = 1;

%
% Step 1: Move speaker c from Ci to Bi until Bprime = B
%

  while Bprime < B,

    tmp2 = [0];

    for cloop = 1 : length(Ci),

      tmp = [0];

      for bloop = 1 : length(Bi),

        tmp(bloop) = matrix( Bi(bloop), Ci(cloop) ) - matrix( spkrnum, ...
          Ci(cloop) );
      end; % for bloop = 1 : length(Bi)
      tmp2(cloop) = sum(tmp) / Bprime;
    end; % for cloop = 1 : length(Ci),
    Bprime = Bprime + 1;
    Nc = Nc - 1;
    % Find the largest c and move from Ci to Bi
    [tmp3, tmpindex] = sort(tmp2);
    Bi(Bprime) = Ci( tmpindex( length(tmpindex) ) );
    Ci = nixcol( Ci,  tmpindex( length(tmpindex) ) );
  % Get another background speaker from Ci until Bprime = B
  end; %   while Bprime < B,


%
% fndfchrt.m
%
% [cohorts] = fndfchrt(distortionmatrix, N, B, spkrnum);
%
% fndfchrt finds the far cohorts for a given distortion matrix.
%
% Inputs:
%    distortionmatrix   a square matrix of distortion scores
%    N          size of pools to use in selecting background speakers
%    B          size of background speakers for maximally spread close
%    spkrnum  indicates number of speaker in distortionmatrix
%
% Ouput:
%    Bi  vector of pointers to speakers determined to be cohorts
%
% References:
%    Reynolds "Speaker Identification and verification using
%              Gaussian mixture speaker models." Speech Communication
%              17 (1995) p 91-108
%
% Created by Capt R. Brian Reid
%
% Created on 14 Aug 1997
%
% Last Modified:  26 Aug 1997
%                  1138
%
function Bi = fndfchrt(matrix, N, B, spkrnum);

Nc = N;
Nf = N;

numspkrs = size(matrix,2);

% Select spkrnum's utterance against all models
Dv = matrix(spkrnum, : );

% Create an index of all of the speakers
indexptr = linspace( 1, numspkrs, numspkrs);

% Remove current speaker from the list
Dv = nixcol(Dv, spkrnum);
```

```
indexptr = nixcol(indexptr, spkrnum);

% Sort Dv and get an index back
[Dvnew, index] = sort(Dv);

% Reorganize indexptr according to index
indexptrnew = indexptr(index);


% Select N further speakers (speakers with greatest distortion)

bsize = length(Bi);

Fi = [ indexptrnew( length( indexptrnew ) - Nf : length( indexptrnew ) ) ];

%
% Step 0: Move the furthest speaker from Fi to Bi
%

 Bprime = 1;
 Bi(1) = Fi(Nf);
 Fi = nixcol(Fi, Nf );
 Nf = Nf - 1;

%
% Step 1: Move speaker f from Fi to Bi until Bprime = B
%

 while Bprime < B,

   tmp2 = [0];

   for cloop = 1 : length(Fi),

     tmp = [0];

     for bloop = 1 : length(Bi),

       tmp(bloop) = matrix( Bi(bloop), Fi(cloop) ) * matrix( spkrnum, ...
           Fi(cloop) );

     end; % for bloop = 1 : length(Bi)

     tmp2(cloop) = sum(tmp) / Bprime;

   end; % for cloop = 1 : length(Fi),

   Bprime = Bprime + 1;
   Nf = Nf - 1;

   % Find the largest f

   [tmp3, tmpindex] = sort(tmp2);

   Bi(Bprime) = Fi( tmpindex( 1 ) );

   Fi = nixcol( Fi, tmpindex( 1 ) );

 % Get another background speaker from Fi until Bprime = B

 end; % while Bprime < B,

%
% fndindx.m
%
% [indices] = fndindx(shortlist, fulllist)
%
% Find the indices of a subset from a complete list
%
```

```
function [indices] = fndindx(shortlist, fulllist)

indices = [];

for loop = 1: size(shortlist,1),
  for loop2 = 1 : size(fulllist,1),

    if fulllist(loop2,:) == shortlist(loop,:),
      indices(loop) = loop2;
    end;

  end; % loop2 = 1 : length(fulllist),

end; % for loop = 1: length(shortlist),


%
% loadphndb.m
%
% function phn=loadphndb;
%
% Description: This function reads in a tabular "data base" of all phoneme
%              labels and their classification (voiced/unvoiced) allowed in
%              the TIMIT phoneme files.
%
% Author: Capt Al Arb, USAF
% Date: 29 Jul 96
% Modified:
%
% Input parameters:
%      none
%
% Output Parameters:
%     phn: A matrix containing each possible phoneme label (columns 1-4) and
%          its classification ("VOICED" or "UNVOICED") (columns 5-13).
%
% Subroutines directly called:
%     none
%
% Subroutines indirectly called:
%     none
%
function phn=loadphndb;

%
% Save current directory location so we can return here.
%
chgdir=pwd;
chgdir=['cd ' chgdir];
%
% Go to location of "data base" file.
%

%
% Open file
%
fid=fopen('/home/hawkeye5/96d/harb/matlab/thesis_code/timit_phoneme.txt','r');
%
% Read in data base as a long string of characters
%
p=setstr(fread(fid,'char'));
%
% Since each row/phoneme is 23 characters wide and there are 62 possible
% labels, reshape into a 62x23 matrix.
%
p=reshape(p,23,62)';
%
```

```
% Save the label (columns 1-4) and V/UV label (15-23).
%
phn=[p(:,1:4) p(:,15:23)];
%
% Close the file
%
fclose(fid);

%
%  maincohort.m
%
%  Read in a list of speakers' scores for a given speaker's utterance
%

%  Set up

close all
clear all

N = 20;  %  Size of pool to use in selecting background speakers

Bt = 10;  %  Total number of background speakers to use as close
          %  and far cohorts.  Should be an even number.

corpus = ['timitmw'];

B = floor(Bt/2);  %  Number of far or close cohorts to pick

%  load a list of speakers

    % For actual
    % speaklist = ['/home/fuggles1/rreid/speakerlist/',corpus,'/allspkr.lis'];
    speaklist = ['/home/fuggles1/rreid/speakerlist/timit/testspeaker.lis'];
    speakers = readafil(speaklist);

numspeakers = length(speakers);

% cohortsdir = ['/home/fuggles1/rreid/toy/Cohorts'];
 cohortsdir = ['/home/fuggles1/rreid/Cohorts/', corpus];

%  For each speaker in the list
starttime = cputime;

for loop = 1 : numspeakers,

  %  Read in the list of scores for speakers' models for speaker(loop)'s sa1

  oscorefile = [cohortsdir '/' speakers(loop,:) '/sa1scores' ];

  oscores = readfil2(oscorefile);

  %  Read in the list of scores for speakers' models for speaker(loop)'s sa2

  sscorefile = [cohortsdir '/' speakers(loop,:) '/sa2scores' ];

  spkr1scores = readfil2(sscorefile);

  %  Put scores into the proper matrix

  sa1matrix( loop, : ) = oscores';

  sa2matrix( loop, : ) = spkr1scores';

end;  % for loop = 1 : numspeakers,

stoptime = cputime;

samatrixtime = stoptime - starttime;

%  Save the sa1 and sa2 matrices for future use (just in case)

eval (['save ' cohortsdir '/samatrix sa1matrix sa2matrix '...
    'samatrixtime' ])
disp(['Determined SA matrices'])

% Determine distortion metric scores
starttime = cputime;
[distortionmatrix] = distmtrx(sa1matrix, sa2matrix);
stoptime = cputime;
% Save the distortion scores
distortiontime = stoptime - starttime;
eval(['save ' cohortsdir '/dismtrx  distortionmatrix distortiontime' ]);

% Find the cohorts
starttime = cputime;
for loop = 1 : numspeakers,
  ccohorts = fndcchrt(distortionmatrix, N, B, loop);
  fcohorts = fndfchrt(distortionmatrix, N, B, loop);
  cohorts = [ccohorts, fcohorts];
  ncohorts(loop,:) = cohorts;
  % Save the cohorts for future use
  cohorts = cohorts';
end;  % for loop = 1 : numspeakers,
stoptime = cputime;
cohorttime = stoptime - starttime;
% Save the entire corpus' cohorts as a single matrix
eval(['save ' cohortsdir '/cohorts  ncohorts'])
eval(['save ' cohortsdir '/cohorttime  cohorttime'])

%
% modwmfcc.m
%
% This m-file creates MFCCs by first warping the utterance according to a
% given speaker's average frequency spectrum (the "Modified Wiener" approach).
%

clear all

eaddr = ['rreid@hawkeye.afit.af.mil'];

corpus = ['timit'];
version = ['mw'];

% Which type original db, or parameter
typefile = ['orig'];

emsg = [' MFCCs made '];
% emsg = [' labelled segments made from ' corpus ' '];

uorv = ['u'];
uorv = ['v'];

% !Hcopy -C configuration_file input_file  output_file
configfile = ['/home/fuggles1/rreid/htkscripts/timithconfig'];
```

```
hcopy1 = ['!HCopy -C ', configfile ];

basedir = ['/home/fuggles2'];

savedirv = ['/home/fuggles1/rreid/mfcc/', corpus, version ];

msrcdir = ['/home/fuggles1/rreid/uttlists/', corpus, '/', typefile];

modeldir = ['/home/fuggles1/rreid/FFTmodels'];

regions = [1, 2, 3, 4, 5, 6, 7, 8];

regions = [ 2];

% Set flags
%
% Set flag for whether to make the unvoiced portions as well as the voiced ones.
%
unvoicedflag = 0;
voicedflag = 1;

testonly = ['y'];

samplefrequency = 16000;
windowsize = 0.020;
fftsize = 512;

sf = samplefrequency;  % Sample frequency
wlength = windowsize;  % window length
fstep = [0.010];  % frame step size
param = 0;  %  Setting based on HTK formats 0 = wav
phnfile = [];  % dir with name of file (*.phn) with phoneme labels

% Note:  ['00000'] does not work properly for speakerold
speakerold = ['zzzzz'];

%  Set trotst to 1 when pulling raw data from the test regions
trotst = 1;

while trotst <= 2,

    if trotst == 2
        setdir = ['train'];
    else
        setdir = ['test'];
    end

    regloop = 1;

    for  regloop = 1 : length(regions),

        tmpdir =['/home/fuggles1/rreid/tmpr', num2str( regions(regloop) ) ];

        regdir = ['dr', num2str( regions(regloop) )  ];

        usedir =[basedir, '/', corpus, '/' , setdir, '/', regdir ];

        getfile = [msrcdir, '/', regdir, setdir, 'list.txt'];

        %  Open the list of utterances

        [fid,message]=fopen(getfile,'r');

        %  Read in the lines of getfile until reaching end of file

        done=0;

        while ~done

          %  P is the utterance to use
          P=fgetl(fid);
```

```
%    disp('accomplished fgetl')

if ~isstr(P)

  % Not a string so set done to quit
  done=1;

else

  % P is a valid string so continue
  % Create a string to use for storage

  p2 = fliplr(P);     % Reverse the sequence

  % Get utterance and extension only and put in normal order

  utterance = fliplr( p2( 1 : find(p2=='/') - 1 ) );

  % Get just utterance name  the extension
  utterance2 = utterance( 1 : find( utterance == '.' ) - 1);

  region = regdir;

  % Next line assumes data is in NIST format of
  % corpus/section/region/speaker/utterance
  % and speaker names are five (5) characters long

  speaker = P( length(P)-(length(utterance) + 5) : ...
     length(P)-(length(utterance) + 1) );

  usefile = [ tmpdir, '/', utterance2, '.swa' ];

  % To pull from the actual data

  usefile2 = [ usedir, '/', speaker, '/', utterance ];

  % To pull from locations other than actual data
  % usefile2 = [ usedir, '/', speaker, '/', corpus, '/',
  % utterance ];

  temp = [tmpdir, '/', utterance2 ];

  % Remove the header information in order to process in MatLab

  % Remove the NIST header information and byte swap the file
  eval ([ '!bhd ' usefile2 ' | dd conv=swab  of=' temp '.swa' ])

  % Read in the file for use
  temp2 = [temp '.swa'];
  data = read_dat(temp2,'short');

  numsamples = max(size(data));

  % Load in the FFT model
  if speaker ~= speakerold
    % Loads speaker's model as avgfft
    eval(['load ' modeldir '/' speaker ]);
    speakerold = speaker;
    disp(speaker)
  end;

  % Warp the utterance according to the appropriate
  % speaker's spectra

  % Calculate the average DFT of the utterance
  [uttavgfft] = fftavgf(data, fftsize, samplefrequency, ...
     windowsize);

  % Compute the Modified Wiener impulse response
  immw= avgfft ./ uttavgfft;
```

```
            htmmw = fftshift( real ( ifft( immw, fftsize) ) );

            % Convolve the original utterance with the Modified Wiener
            % impluse response (which hopefully diminishes the channel
            % effects)
            newdatammw = conv(data, htmmw);

            % Extract the voiced and unvoiced portions

            phnfile = [ usedir, '/', speaker, '/', utterance2, '.phn' ];

            [voiced,unvoiced]= exvu4htk(newdatammw,usefile,sf,wlength,fstep,phnfile);

            % Write the voiced part to a file

            if (voicedflag == 1),
              temp4 = [tmpdir, '/',  utterance2, '.htk'];
              w_error = whtkwav(voiced, temp4, sf, param);

              if w_error ~= 0,
                disp(['error writing ', temp4])
              % else
                % disp(['OK'])
              end;

            end; % if (voicedflag == 1),

            % Write the unvoiced part to a file

             if (unvoicedflag == 1),
               temp4 = [savediru, '/', speaker, '/', utterance2, '.htk'];
               w_error = whtkwav(unvoiced, temp4, sf, param)

             end; % if (unvoicedflag == 1),

            % Calculate the MFCCs
            hcopy = [hcopy1, ' ', temp4, ' ', savedirv, '/', speaker, '/', ...
                 utterance2, '.mfc'];

            eval(hcopy)

            % Clean up temporary files

            eval(['!rm ', tmpdir, '/', utterance2, '*.*' ] )

          end; % end if ~isstr(P)

        end;     % end while ~done

        emsg2 = ['dr ', num2str(regions(regloop)), ' ', emsg, ' for ', ...
             setdir];

        emailmsg(eaddr,emsg2);

        fclose(fid);

      end;     % end for regloop

      trotst = trotst + 1;

      if testonly == ['y']
        trotst = 3;
      end;

end;     %     while trotst <= 2,

disp(['Done'])

%
```

```
% read_dat.m
%
% [outvect, count]=read_dat(infile,datatype);
%
% Input:
%     infile -  filename of file to read in
%     datatype - format of data, e.g., char or long
%
% Output:
%     outvect - vector of values from infile
%     count  -  number of elements in outvect
%
function [outvect, count]=read_dat(infile,datatype);

  [fid,message] = fopen(infile,'r');

  [outvect,count] = fread(fid,datatype);

  fclose(fid);

%
%  readafil.m
%
%  [list] = readafil(infile)
%
%  Read in a text file and convert its contents into
%       a MATLAB variable
%

function [list] = readafil(infile)

[fid] = fopen(infile, 'r');

done = 0;
list = [];
count = 0;

while ~done
  temp = fgetl(fid);
  if ( isstr(temp) ),
    count = count + 1;
    % list = [list; temp];
    list(count, : ) = [temp];
  else
    done = 1;
  end;
end;

fclose(fid);

%
%  readfil2.m
%
%  [list] = readfil2(infile)
%
%  Read in 2nd column of a text file and convert
%       to a MATLAB variable.  Assumes 7 characters
%       before the first character of second column
%

function [list] = readfil2(infile)

[fid] = fopen(infile, 'r');

done = 0;
list = [];
count = 0;
```

```
while ~done
  temp = fgetl(fid);
  if ( isstr(temp) ),
    count = count + 1;
    % list = [list; temp];
    list(count, : ) = [str2num(temp( : , 7:length(temp) ) ) ];
  else
    done = 1;
  end;
end;

fclose(fid);


%
% readhtkn.m
%
% [data, numsamples] = readhtkn(filename,htkformat);
%
% Read HTK 2.0 files into MatLab.
%
% Potential Bug:
%         This was designed to read HTK waveform files.
%         For other types of files, some adjustments of the
%         fread size may need to be made.
%
% Created by Capt R. Brian Reid
%            19 Aug 1997
%
% Modified 16 Sep 1997 to return the number of samples

function [data, numsamples] = readhtkn(filename,htkformat)

[fid, errmsg] = fopen(filename, 'r');

% errmsg

% Read in the head information

numsamples = fread(fid,1,'int32')
sampleperiod = fread(fid,1,'int32')
samplesize = fread(fid,1,'int16')
paramkind = fread(fid,1,'int16')

if htkformat == 0,
  datatype = ['int16'];
else
  datatype = ['float32'];
end;

% disp(['datatype is ' datatype ])

% Read in the actual data

data = fread(fid, numsamples, datatype);

% Close the file

fclose(fid);


% readphn.m
%
% function [a,b]=readphn(phnfile);
%
% Description: This function reads in the hand labelled TIMIT phoneme file and
%              returns a matrix of phonmeme labels and a matrix of phoneme
%              start points and end points.
%
```

```
% Author: Capt Al Arb, USAF
% Date: 29 Jul 96
% Modified:
%
% Input parameters:
%      phnfile: The name of the TIMIT phoneme file.
%
% Output Parameters:
%     a: A matrix containing the starting point and ending points of each
%        phoneme in columns 1 and 2 respectively.  Each row is a different
%        phoneme.
%
%     b: A matrix of phoneme labels.  Each row is a different phoneme.
%
% Subroutines directly called:
%     none
%
% Subroutines indirectly called:
%     none
%
function [a,b]=readphn(phnfile);

a=[];
b=[];
%
% Open TIMIT phoneme file for reading.
%
[fidphn, phnmsg] = fopen(phnfile,'r');

% phnmsg

%
% Continue to read until reaching the end-of-file.
%
while ~feof(fidphn)
  %
  % Get one line of the file as a string.
  %
  s=fgetl(fidphn);

  %
  % Check to see if it's the end of file, if not, continue to process.
  %
  if s~=(-1)
    %
    % Break up string into 2 integers and a string.
    %
    p=sscanf(s,'%i %i %s');

    %
    % The two integers are the start point and end point of the phoneme.
    %
    a=[a;p(1) p(2)];

    %
    % Set the numerical version of the label string to an actual string.
    %
    p=setstr(p(3:length(p)))';

    %
    % Prepend the string with spaces to bring length of string to 4 with the
    % label right justified.
    %
    if length(p)==2
        p=['  ' p];
    elseif length(p)==3
```

```
      p=['  ' p];
    elseif length(p)==1
      p=['   ' p];
    end;

    %
    % add new label to b matrix
    %
    b=[b;p];

  end;
end;
fclose(fidphn);

%
%  spkrid.m
%
%  This file pulls in utterance scores for all speakers and determines the
%  identity of the speaker based on highest score by GMM model.
%
clear all

corpus = ['timit'];
version = ['cmsbl'];

saveflag = ['y'];

tgtfile = ['/home/fuggles1/rreid/Results/' corpus version '/idsanew'];
idfile = ['/home/fuggles1/rreid/Results/' corpus version '/idsa'];


cohorts = [];


% Get the needed info

% for actual
speaklist = ['/home/fuggles1/rreid/speakerlist/', corpus, '/alltsttr.lis'];

eval(['load /home/fuggles1/rreid/Cohorts/timit' version '/cohorts']);

speakers = readafil(speaklist);
numspeakers = length(speakers);

ncohorts = cohorts;

% Zero out the confusion matrix
cmatrix = zeros(numspeakers);

% Read in a list of files to check

% Get the scores and put into the proper form

file2read = ['/home/fuggles1/rreid/Results/' corpus version '/sascores.txt'];

fid = fopen(file2read,'r');

done = 0;
correct = 0;
errors = 0;
count = 1;
spkrnum = 1;
filenum = 1;

while ~done
  nowfile = fgetl(fid);

  if isstr(nowfile)

    % Load in scores for a given utterances
    % Use for MatLab files
    % eval(['load ' nowfile ]);
    % scores = sxscores;

    % Use for ASCII files
     scores = readfil2(nowfile)';
     lscores = max(size(scores) );
     scores = scores( lscores - numspeakers + 1 : lscores);

    % Determine the winner

    [tmp, tmpindex] = sort( scores );
    winner = tmpindex(length(tmpindex) );
    temp =  fliplr( nowfile );
    temp1 = find( temp == '/' );

    realspkr = fliplr( temp(temp1(1) + 1 : temp1(2) - 1 ) );

    realspkrnum = fndindx(realspkr, speakers);

    % Update the confusion matrix based on which speaker made the utterance
    % and which speaker had the highest score for the utterance

    cmatrix(realspkrnum, winner) = cmatrix(realspkrnum, winner) + 1;

    if realspkrnum == winner
      correct = correct + 1;
    else
      errors = errors + 1;
    end;

    % disp(['Finshed file ' num2str(filenum) ])

    filenum = filenum + 1;

  else
    done = 1;

  end;  % if isstr(nowfile);

  % Get the next utterance
end;  % while ~done

fclose(fid)

correct
errors

total = correct + errors

percorrect = correct/total*100

pererror = errors/total*100

correct2 = sum(diag(cmatrix))

% save the identification results

if saveflag == ['y']
   eval(['save ' idfile ])
end;


%
% verifhtk2.m
%
% [score] = verifhtk2(speaker, cohorts, speakerlist, uttscores)
%
% Script to perform verfication of a for a given utterance using utterances
% already calculated by HTK 2.0
```

```
%
% Inputs:
%    speaker  number of speaker as determined by speakerlist
%    cohorts  vector of cohorts containing integer values of cohorts in
%    speakerlist
%    speakerlist  list of all speakers
%    uttscores    scores for all models for a given utterance
%
% Output:
%    score   normalized log probability score of speaker given cohorts
%

function [score] = verifhtk2(speaker, cohorts, speakerlist, uttscores)

% Calculate the score for speaker

oscores(1) = uttscores(speaker);

% Calculate the scores for the cohorts and append them to the file
% containing the speaker's score

for cloop = 1 : length(cohorts),

  oscores(cloop + 1) = uttscores(cohorts(cloop) );

end;

% Determine the actual score based on score = speakerscore -
% sum(cohortscores)/#cohorts

score = oscores(1) - sum(oscores( 2 : length(oscores) ) ) / length(cohorts);

% end of verify


%
% whtkwav.m
%
% function w_error = whtkwav(data,filename,sampleperiod,param)
%
% Writes waveform data to 'filename' in HTK standard binary format.
% The data is written with the appropriate 12 byte header together with
% the data in the proper byte format.
%
% Ensure the data is passed as a matrix with each row corresponding
% to a frame, and each column contains the parameter (fft spectra, etc)
%
% Originally from Al Harb's:
%       function w_error = write_HTK_param(data,filename)
%
% Modifications allow:
%    Varying sample periods
%    writing back into the desired parameter format
%       Use 0 for waveform 6 for HTK MFCC and 9 for user defined
%       Reference HTK V2.0 manual page 73
%
% Modifications by R. Brian Reid
% Modified: 20 Aug 1997 1350
%
function w_error = whtkwav(data, filename, samplefreq, param)

fid = fopen(filename,'w');
if (fid  == -1);
  error('Unable to open the file to write HTK paramter data');
end

% Check the number of input arguments
if nargin < 3,
  % Use the defaults
  samplefreq = 16000;
  param = 0;
end;  % if nargin < 3

% Compute sample period from sample frequency in terms of 100ns

sampleperiod = ceil( 1 / samplefreq / ( 100 * 10^(-9) ) );

% Determine the number of bytes per sample

if param == 0,
  bytespersamp = 2;
  datasize = ['int16'];
  numsamples = length(data);
else
  bytespersamp = 4*size(data,2);
  datasize = ['float32'];
  numsamples = size(data,2);
end;

% Write the header

% Write the number of samples in the file (4-byte header)
fwrite(fid,numsamples,'int32');

% Write the sample period in 100ns units (4-byte integer)
fwrite(fid,sampleperiod,'int32');

% Write the number of bytes per sample
% need a 4-byte float for each paramter in the feature vector
fwrite(fid,bytespersamp,'int16');

% Write code indicating the sample kind (2-byte integer)
%     Use 0 for HTK waveform, 6 for HTK MFCC, and 9 for user defined
fwrite(fid,param,'int16'); % User defined parameters data flag

% Write the data into file in datasize chunks
fwrite(fid,data,datasize);

fclose(fid);
w_error = 0;
return;
```

B.2 C-Shell Scripts [2]

```
#!/bin/csh
#  gmm2maker
#  UNIX C-shell script for creating Gaussian Mixture Models (GMM) using HTK 2.0
#
#   Currently set for full TIMIT
##
#  Output  Files of HMMs for each speaker in the speaker list
#
#  Assumptions:  Using voiced speech (only in terms of file locations)
#

set corpus = timit

set version = cmsbl

set eaddr1 = "rreid@hawkeye.afit.af.mil"
set eaddr2 = "rreid"

set emsg = "GMMs have been created for $corpus region $1"
```

[2] *Lines beginning with* - S *should be made into a continuation of the preceeding line.*

```
set hconfig = /home/fuggles1/rreid/htkscripts/hconfig$version
set mlf = /home/fuggles1/rreid/LABEL/tmaster.mlf
set spkrlab = /home/fuggles1/rreid/toy
set tmpdir = "/home/hawkeye12/97d/rreid/SID/SID"
cd $tmpdir
set spkrlist = /home/fuggles1/rreid/speakerlist/$corpus/trainspeaker{$1}.lis
set hmmdir = "/home/fuggles1/rreid/tmpr"{$1}
# Source directory for lists of utterances for each speaker
set srcdir = /home/fuggles1/rreid/uttlists/{$corpus}{$version}
set mfcctgtdir = /home/fuggles1/rreid/hmm
set mfcctgtdir1 = $mfcctgtdir/$corpus$version
if (! -d $mfcctgtdir1 ) then
    mkdir $mfcctgtdir1
    chmod 775 $mfcctgtdir1
endif
foreach speaker (`cat $spkrlist`)
    set mfcctgt = $mfcctgtdir1"/"$speaker
    set trainfile = "$srcdir/$speaker/sasisx3.tra"
    echo $speaker > $hmmdir/hmmlist
    echo $speaker
    echo "MU 2 {$speaker.state[2].mix}" > $hmmdir/edcdlis1
if (! -d $hmmdir/hmm.0 ) then
   mkdir $hmmdir/hmm.0
    chmod 774 $hmmdir/hmm.0
endif
if ( -d $hmmdir/hmm.1 ) then
   rm -fr $hmmdir/hmm.1
endif
mkdir $hmmdir/hmm.1
if ( ! -d tmp ) mkdir tmp

if ( -e $hmmdir/hmm.1/$speaker ) then
   rm -f $hmmdir/hmm.1/$speaker
endif
if ( -e $hmmdir/hmm.0/$speaker ) then
   rm -f $hmmdir/hmm.0/$speaker
endif

#HInitHInit -C $hconfig -i 15 -L $spkrlab -v 0.01 -o $speaker -M $hmmdir/hmm.0
      #    -S $trainfile protogmm5
HInit -C $hconfig -i 15 -L $spkrlab -v 0.01 -o $speaker -M $hmmdir/hmm.0
      -S $trainfile protogmmcms
HRest -C $hconfig -i 15 -L $spkrlab  -v 0.01 -M $hmmdir/hmm.1
      -S $trainfile $hmmdir/hmm.0/$speaker
cp $hmmdir/hmm.1/$speaker $hmmdir/hmm.0/$speaker
```

```
HHEd -C $hconfig -d $hmmdir/hmm.0 -M $hmmdir/hmm.1 $hmmdir/edcdlisl  $hmmdir/hmmlist

cp $hmmdir/hmm.1/$speaker $hmmdir/hmm.0/$speaker

@ maxNum = 32
@ numiter = (${maxNum} - 2 ) / 2
@ loop = 1
@ index = 2

while ( ${loop} <= {$numiter} )
    @ index = $index + 2

    set mixturecmd="MU $index "

    echo $mixturecmd  "{$speaker.state[2].mix}" > $hmmdir/edcmd

    echo $mixturecmd

   HHEd -C $hconfig -d $hmmdir/hmm.0 -M $hmmdir/hmm.1 $hmmdir/edcmd $hmmdir/hmmlist

    cp $hmmdir/hmm.1/$speaker $hmmdir/hmm.0/$speaker

    HRest -C $hconfig -i 15 -L $spkrℓab  -v 0.01 -M $hmmdir/hmm.1
         -S $trainfiℓe $hmmdir/hmm.0/$speaker

    cp $hmmdir/hmm.1/$speaker $hmmdir/hmm.0/$speaker

    @ loop ++
end

cp $hmmdir/hmm.1/$speaker ${mfcctgt}
chmod 774 ${mfcctgt}

#  loop to get the next speaker

rm  $hmmdir/hmm.0/$speaker
rm  $hmmdir/hmm.1/$speaker

end

#  Release the HTK license

Hfree

#  Notify user that GMMs have been created

cd ~rreid/matlab/thesis/tools

mailer.c  $eaddrl $emsg
```

```
# !/bin/csh

#  uttscores2.c
#
#  UNIX C-shell script to determine scores for all models
#       for a given utterance using the Viterbi algorithm
#       in HVite.
#
#  Input:  $1 is the region to determine scores for
#
#  Variables to set:
#
#     srcdir      Path for Speaker list, sets Results,
#     srchmm        Path for location of hmms
#     tgtdir      Path to place
#     scriptdir   name of directory containing directories of sa, si, sx, all.tra lists
#     uttdir      Path for actual parametized utterances

set eaddr = rreid@hawkeye.afit.af.mil

set uttfile = sa.tra
set region = {$1}
set corpus = timit
set version = cmsbl
set emsg = "Probability scores calculated for $corpus$version. "
set mastermlf = /home/fuggles1/rreid/LABEL/{$corpus}master.mlf
set uttlistdir = /home/fuggles1/rreid/uttlists/{$corpus}{$version}
set hconfig = /home/fuggles1/rreid/htkscripts/hconfig{$version}

# For final problem
set srcdir = /home/fuggles1/rreid
set srchmm = /home/fuggles1/rreid/hmm/timit$version
set srcdir2 = /home/fuggles1/rreid/uttlists/{$corpus}{$version}
set hmmdir = $srchmm
set tgtdir = /home/fuggles1/rreid/Cohorts/{$corpus}{$version}
set uttdir = /home/fuggles1/rreid/mfcc/{$corpus}{$version}
set speakerlist = /home/fuggles1/rreid/speakerlist/timit/speaker{$region}s.lis
set allspeakerlist = /home/fuggles1/rreid/speakerlist/$corpus/trainspeaker.lis
# Needed for HTK V2.0 (speakerdic and spknet can be the same for all corpi)
set speakerdic = /home/fuggles1/rreid/Networks/timit/timit.dic
set spknet = /home/fuggles1/rreid/Networks/timit
    # Check to ensure the appropriate directory exists
    if (! -d {$tgtdir} ) then
        mkdir {$tgtdir}
        chmod 774 {$tgtdir}
    endif

foreach spk (`cat $speakerlist`)
  # Ensure speakers directory exists
  if (! -d {$tgtdir}/{$spk} ) then
      mkdir {$tgtdir}/{$spk}
      chmod 774 {$tgtdir}/{$spk}
  endif
  set Resultsdir = $tgtdir/$spk
    # Save speaker2 and score for speaker1s utterance
  foreach spk2 (`cat $allspeakerlist`)
    # For each speaker, spk2, get a score for each utterance
    # File name containing current hmm
    printf "%s\n" $spk2 > $hmmdir/hmmlist{$region}
    foreach utterance (`cat $uttlistdir/$spk/$uttfile`)
      set utter = {$utterance}
      set utter2=`basename $utter`
      set utter2=$utter2:r
      set tgtfile = {$utter2}scores
      set results = {$tgtdir}/{$spk}/{$tgtfile}
```

```
        printf "%s\t" $spk2 >> {$resuℓts}
        HVite −C $hconfig −a −d $srchmm −I $mastermℓf −y svd −l $Resuℓtsdir −o N
                −w {$spknet}/{$spk2}.net $speakerdic {$hmmdir}/hmmlist{$region} $utter
        awk '{printf("%s\n",$4);}' {$Resuℓtsdir}/{$utter2}.svd >> {$resuℓts}
        rm {$Resuℓtsdir}/{$utter2}.svd
        # Get next utterance
      end
      # Get next speaker2
    end
  # Get next speaker1
  end
  set emsg2 = "Region $region $emsg"
  cd ~rreid/matlab/thesis/tools
  mailer.c $eaddr $emsg2 $resuℓts
```

Bibliography

1. "The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT): NIST Speech Disc CD1-1.1." Online documentation files, October 1990.

2. "NTIMIT Speech Corpus CD-ROMs (NTIMIT): NIST Speech Discs 10-1.1, 10-2.1." Online documentation files, August 1992.

3. Avendano, Carlos and Hynek Hermansky. "On the Effects of Short-Term Spectrum Smoothing in Channel Normalization," *IEEE Transactions on Speech and Audio Processing* (1997).

4. Bishop, Christopher M. *Neural Networks for Pattern Recognition*, chapter 5. Oxford University Press Inc., 1995.

5. Davis, Steven B. and Paul Mermelstein. "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences," *IEEE Transactions on Acoustics, Speech and Signal Processing* (1980).

6. Gish, Hervert and Michael Schmidt. "Text-Independent Speaker Identification," *IEEE Signal Processing Magazine*, 18–32 (October 1994).

7. Higgins, A. L., et al. "Speaker Verification Using Randomized Phrase Prompting," *Digital Signal Processing*, *1*:89–106 (1991).

8. Lim, Jae S. *Two-Dimensional Signal and Image Processing*. Englewood Cliffs, NJ 07632: PTR Prentice-Hall, Inc., 1990.

9. Mammone, Richard J., et al. "Robust Speaker Recognition: A Feature-based Approach," *IEEE Signal Processing Magazine*, 58–71 (September 1996).

10. Rabiner, Lawrence R. and Biing-Hwang Juang. *Fundamentals of Speech Processing*. 113 Sylvan Avenue, Englewood Cliffs, NJ 07632: PTR Prentice-Hall, Inc., 1993.

11. Reynolds, Douglas A. *A Gaussian Mixture Modeling Approach to Text-Independent Speaker Identification*. PhD dissertation, Georigia Institute of Technology, August 1992.

12. Reynolds, Douglas A. "Experimental Evaluation of Features for Robust Speaker Identification," *IEEE Transactions on Speech and Audio Processing*, *2*(4):639–643 (October 1994).

13. Reynolds, Douglas A. "Automatic Speaker Recognition Using Gaussian Mixture Speaker Models," *The Lincoln Laboratory Journal*, *8*(2):173–191 (1995).

14. Reynolds, Douglas A. "Speaker identification and verfication using Gaussian mixture speaker models," *Speech Communication*, *17*:91–108 (1995).

15. Reynolds, Douglas A., et al. "The Effects of Telephone Transmission Degradations on Speaker Recognition Performance." *ICASSP 1995*. 329–332. 1995.

16. Rosenberg, A. E. "The Use of Cohort Normalized Scores for Speaker Verification." *Proceedings of the International Conference on Spoken Language Processing*, edited by Banff. 599–602. 1992.

17. Thomas G. Stockham, Jr. and Thomas M. Cannon. "Blind Deconvolution Through Digital Signal Processing," *Proceedings of the IEEE* (1975).

18. Young, Steve, et al. *The HTK Book*. Entropic, Cambridge Research Laboratory, 1996.

Vita

Captain Robert Brian Reid was born 25 September 1967 in Topeka, Kansas. He graduated from the University of Kansas in January, 1991, with a Bachelor of Science in Electrical Engineering. Upon graduation, Capt Reid received his commission into the United States Air Force through the Reserve Officer Training Corps.

On 30 September 1991, Capt Reid was assigned to the Penetration Analysis Branch, 544 Intelligence Wing, Strategic Air Command (SAC) at Offutt AFB, Nebraska, as a computer system engineer. With the stand down of SAC in 1992, Capt Reid was reassigned to the 20 Intelligence Squadron's Electronic Intelligence (ELINT) Flight. While in the ELINT Flight, Capt Reid served as computer system engineer, Shift Officer in Charge, and Systems Engineer until May of 1996. While stationed at Offutt AFB, Capt Reid graduated with highest honors from the University of Bellevue with a Master in Business Administration in May 1996.

Capt Reid was assigned to the Air Force Institute of Technology in 1996 where he is currently pursuing a Masters in Electrical Engineering. He is a member of IEEE and Eta Kappa Nu. Upon graduation, Capt Reid will be assigned to the National Air Intelligence Center at Wright-Patterson AFB, Ohio.

Permanent address: 14016 William Circle
Omaha, Nebraska 68144

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** December 1997	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE SPEAKER VERIFICATION IN THE PRESENCE OF CHANNEL MISMATCH USING GAUSSIAN MIXTURE MODELS	**5. FUNDING NUMBERS**
6. AUTHOR(S) Robert Brian Reid	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology, WPAFB OH 45433-6583	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GE/ENG/97D-17
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Capt John Colombi, Ph.D. National Security Agency R221 9800 Savage Road STE 6516 Fort Meade MD 20755-6516	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT Distribution Unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

A channel compensation method is sought for use in speaker identification (ID) and verification applications under matched and mismatched training and testing conditions. This work expands on previous work on matched conditions by investigating three techniques on matched and mismatched conditions using the TIMIT and NTIMIT speech databases. First, previous results on 168 speakers are reproduced for matched conditions using Gaussian mixture models (GMM) and mel-frequency cepstral coefficients. Next, cepstral mean subtraction with band limiting (CMSBL) is investigated. The third method, developed in this thesis, uses a modified Wiener filtering approach to channel compensation. New GMMs are created for each method. The first approach is then expanded to include all 630 TIMIT and NTIMIT speakers for speaker verification. For speaker ID under matched conditions, the CMSBL method had three more errors than no additional preprocessing but yielded the best ID results for the mismatch case with 27.4% correct. Additionally, the CMSBL method yielded the best verification results with an equal error rate of approximately 0.26% for matched conditions on TIMIT and approximately 19.6% for mismatched conditions on NTIMIT.

14. SUBJECT TERMS speaker identification, speaker verification, channel compensation, channel mismatch, TIMIT, NTIMIT, Gaussian mixture models	**15. NUMBER OF PAGES** 68
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL